Itaú Unibanco

Itaú

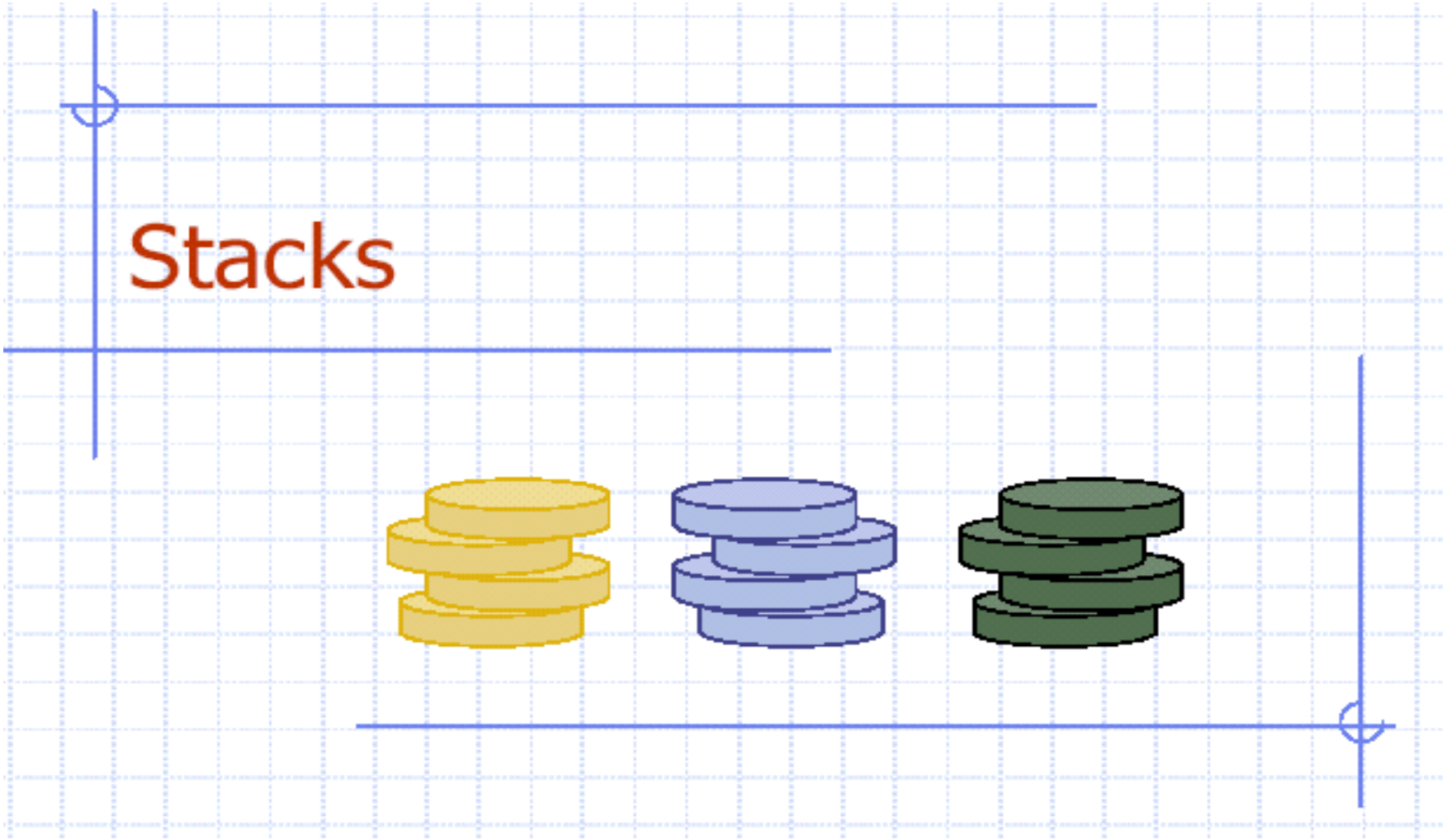
Programa de formação

ITAú analytics

Módulo I – Fundamentos Computacionais
Sessão 6 - Aula 2 - Pilha

Prof. Dr. Luiz Alberto Vieira Dias
Prof. Dr. Lineu Mialaret

Pilhas
Introdução

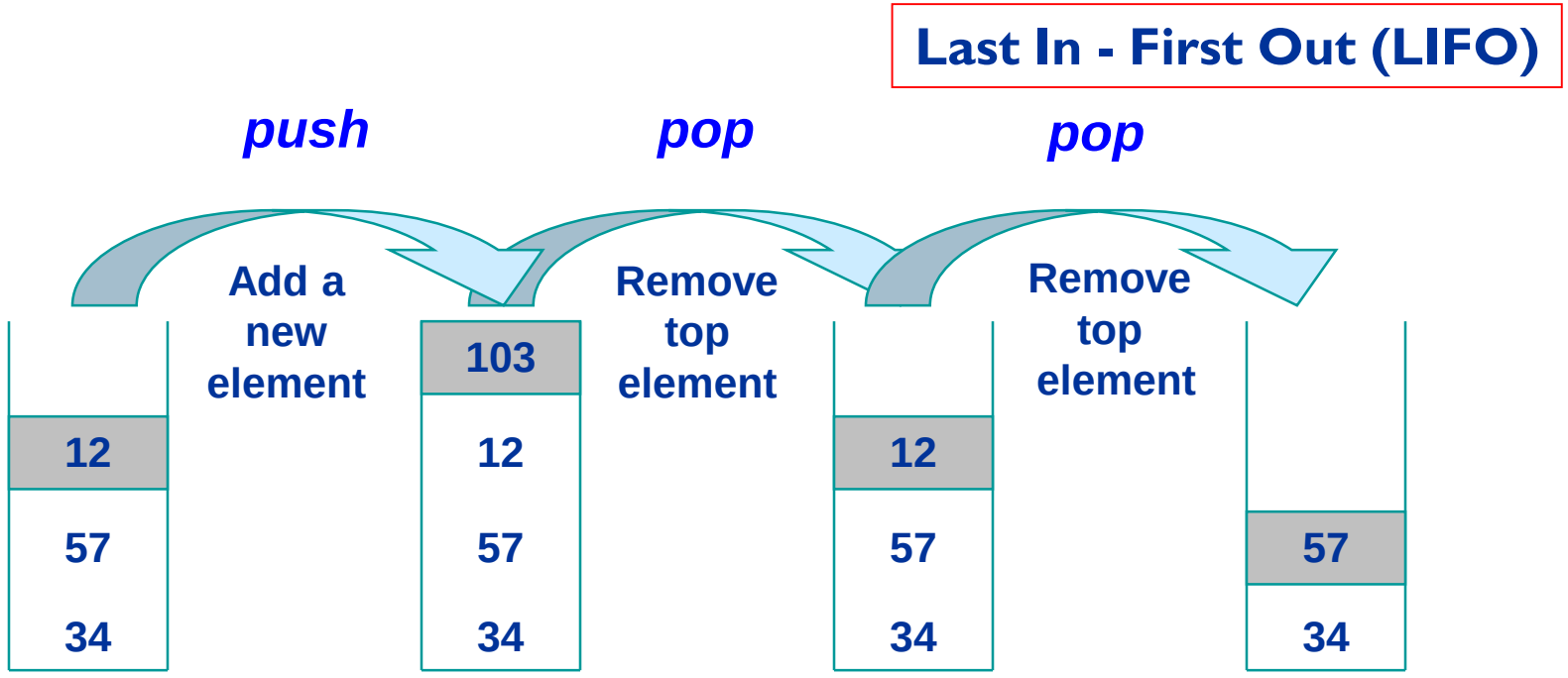


Pilhas
Definição

- Uma pilha é um contêiner de elementos ou objetos ordenados que são inseridos e removidos de acordo com a disciplina de acesso ou princípio chamado ***last-in-first-out*** **(LIFO)**, ou seja, "o último que entra é o primeiro que sai"
- Objetos podem ser inseridos em qualquer tempo, mas somente o último objeto (o mais recentemente inserido) pode ser removido
- A operação de inserir um elemento ou objeto na pilha é conhecida como empilhar ("***pushing***" ou "***push***")
- A operação de desempilhar ("***popping***" ou "***pop***") é o mesmo que remover um elemento ou objeto da pilha

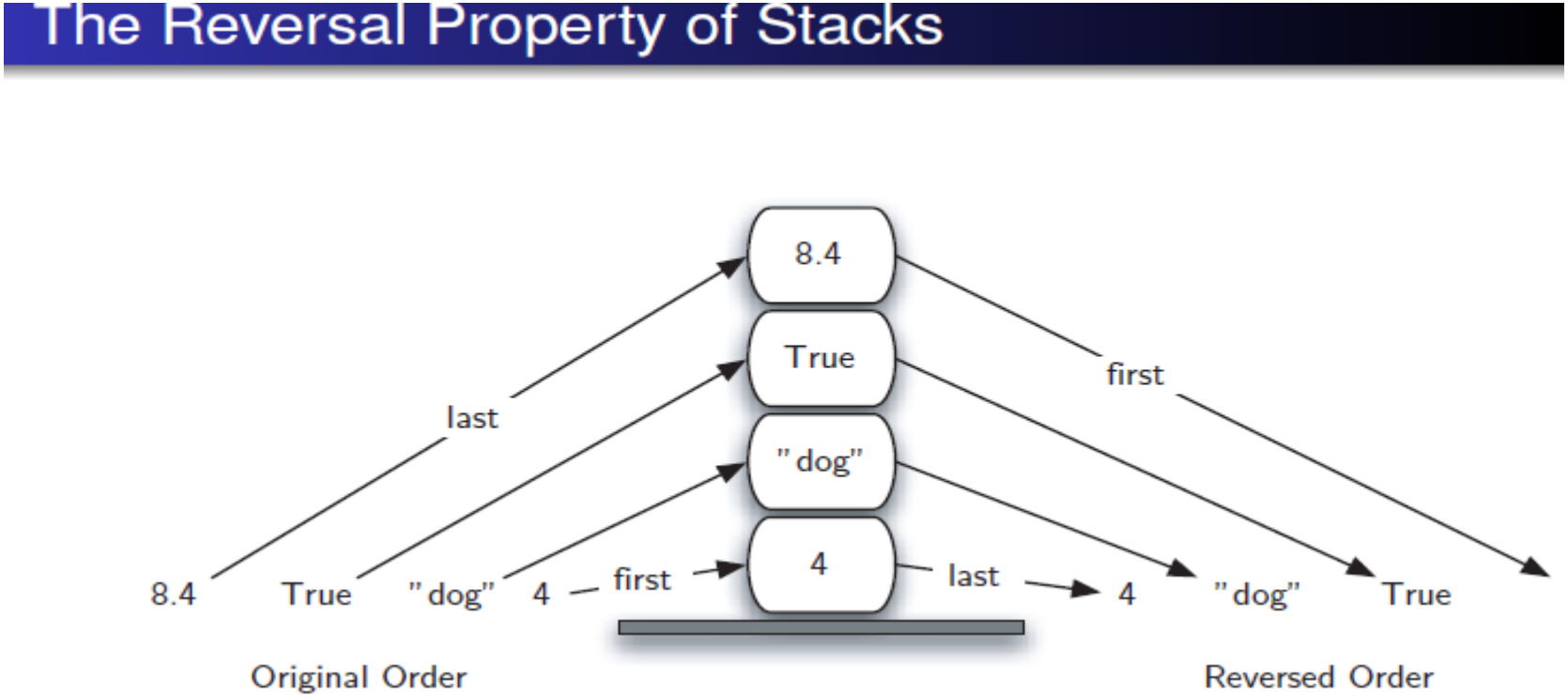
Pilhas
Definição (cont.)

- Add only to the **top** of a Stack
- Remove only from the **top** of the Stack
  - Note: The last item placed on the stack will be the first item removed



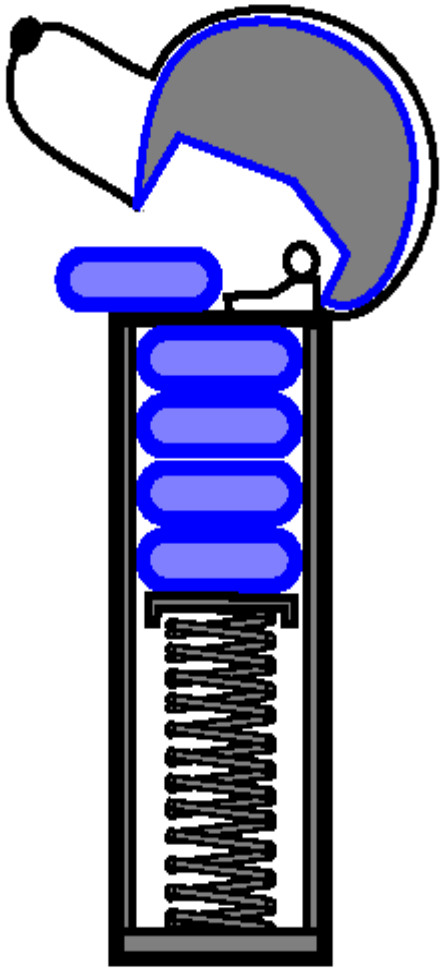
Pilhas
Definição (cont.)

- The **base** of the stack contains the **oldest** item, the one which has been there the **longest**
- For a stack the order in which items are removed is exactly the **reverse of the order** that they were placed



Pilhas
Exemplos Cotidianos

Pilha de pratos em uma cafeteria

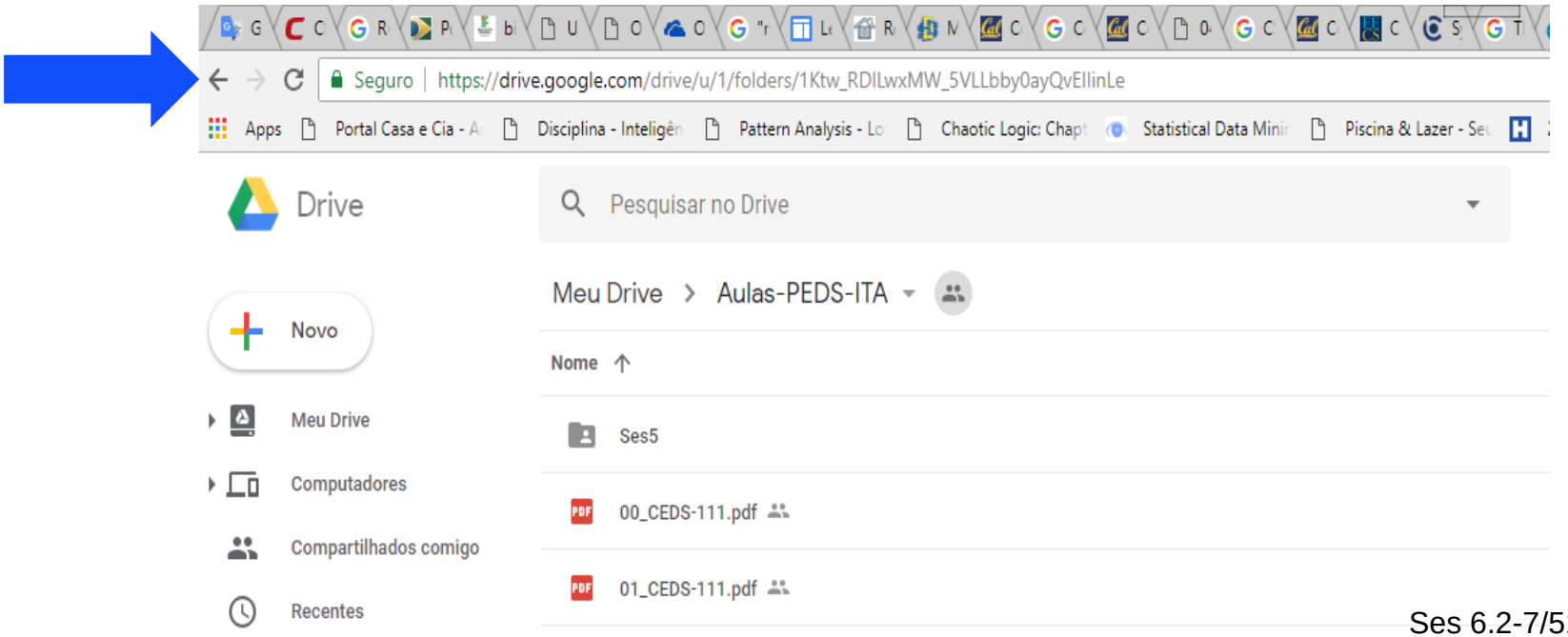


Máquina PEZ de doces implementando fisicamente o tipo de dado pilha

Pilha de dinheiro

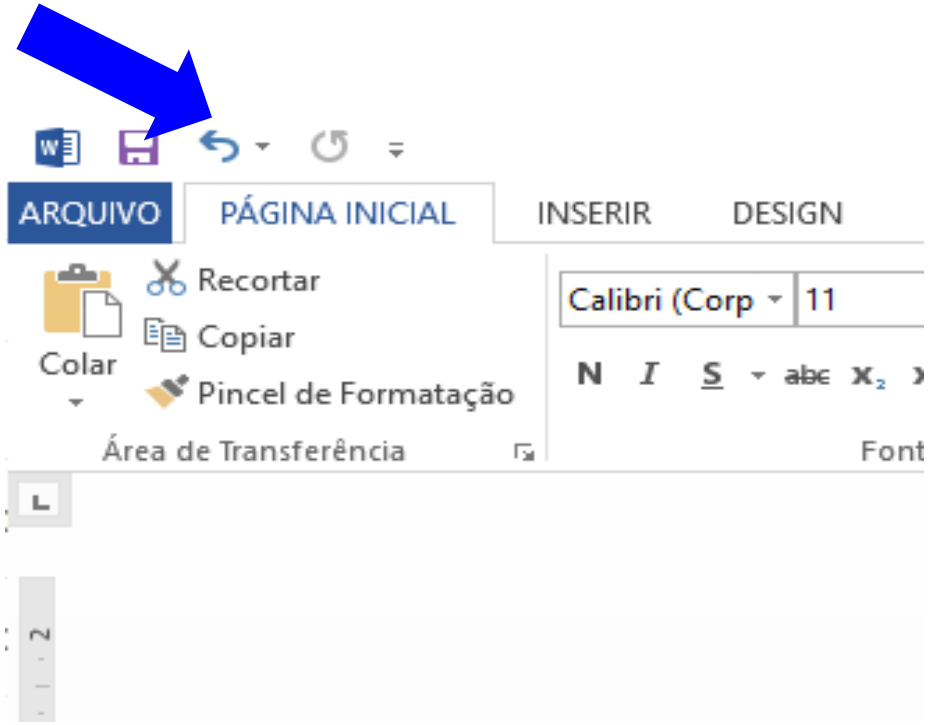
Pilhas
Exemplos de Aplicações

- Pilhas são estruturas de dados fundamentais e são utilizadas em muitas aplicações de software, tais como:
  - Os navegadores (*browsers*) Web, que armazenam os endereços das páginas mais recentemente visitadas numa estrutura de pilha



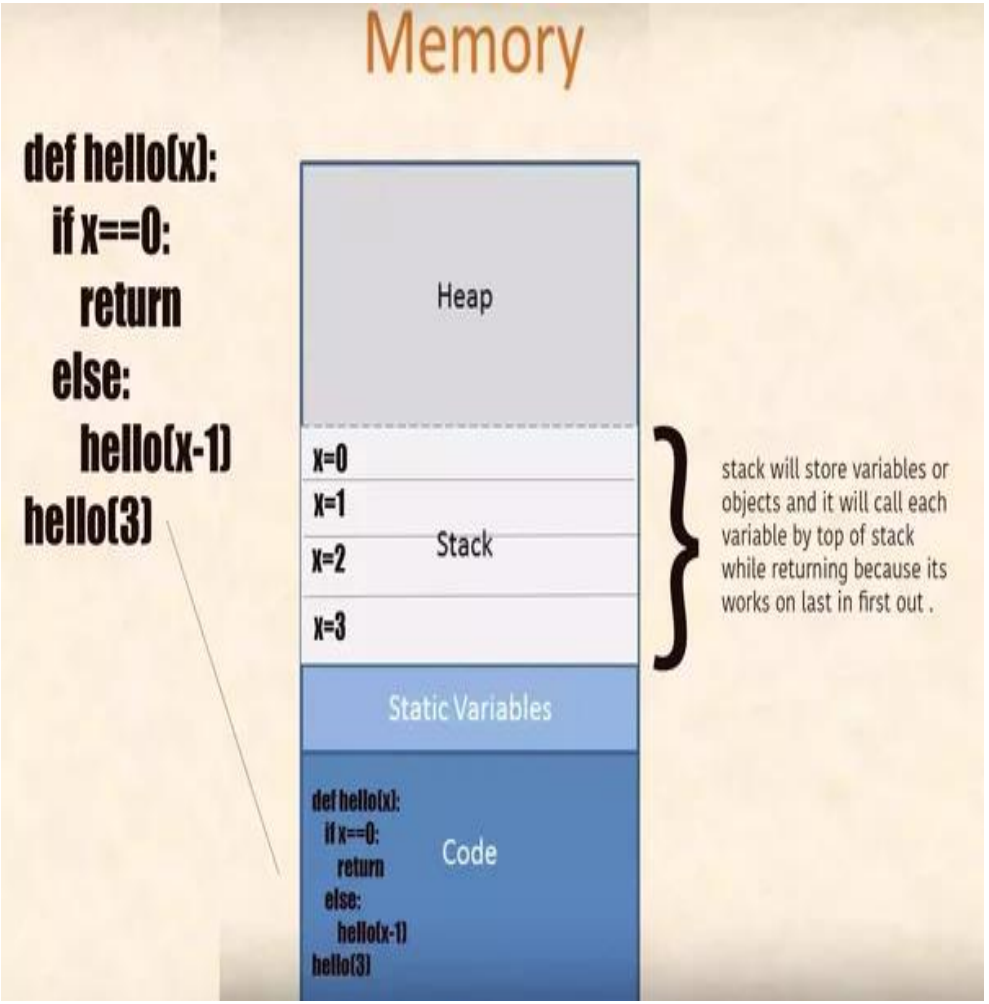
Pilhas
Exemplos de Aplicações (cont.)

- Pilhas são estruturas de dados fundamentais e são utilizadas em muitas aplicações de software, tais como:
  - Os editores de texto, que provêm um mecanismo de "desfazer" (*undo*), normalmente mantêm armazenadas numa estrutura de pilha as mudanças de textos ocorridas



Pilhas
Exemplos de Aplicações (cont.)

- Pilhas são estruturas de dados fundamentais e são utilizadas em muitas aplicações de software, tais como:
  - O encadeamento de chamadas de métodos no Python
  - Estrutura de dados auxiliar para outros algoritmos
  - Componente de outras estruturas de dados



Pilhas
ADT Pilha

- Formalmente, um Tipo Abstrato de Dado (***Abstract Data Type* – ADT**) é um modelo matemático de uma estrutura de dados que especifica os **tipos de dados armazenados**, as **operações suportadas** por eles e **os tipos de parâmetros** das operações
- O ADT especifica o que cada operação faz, mas não como ela é realizada

Pilhas
ADT Pilha (cont.)

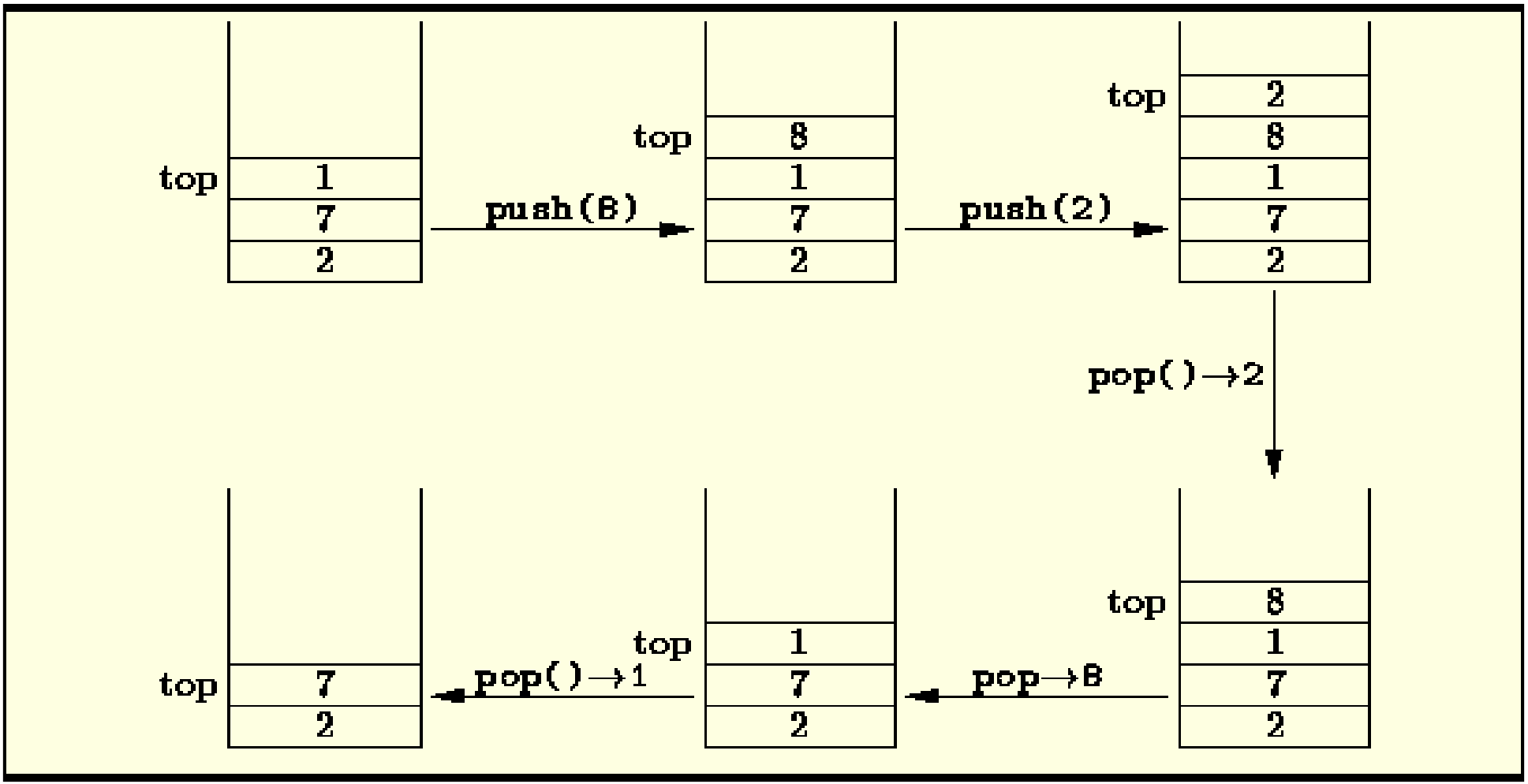
- Uma pilha é um Tipo Abstrato de Dado - ADT que suporta dois métodos principais:
  - `push(o)`: Insere o objeto ***o*** no topo da pilha
    Entrada: objeto; **Saída**: nenhuma
  - `pop()`: Remove o objeto do topo da pilha e retorna-o
    Entrada: nenhuma; **Saída**: objeto
- Os seguintes métodos de suporte também são definidos:
  - `size()`: Retorna o número de objetos na pilha
    Entrada: nenhuma; **Saída**: inteiro
  - `isEmpty()`: Retorna um boolean, indicando se a pilha está vazia
    Entrada: nenhuma; **Saída**: boolean
  - `top()`: Retorna o objeto do topo da pilha, sem removê-lo
    Entrada: nenhuma; **Saída**: objeto

Pilhas
Operações na Pilha

Operation	Return Value	Stack Contents
S.push(5)	–	[5]
S.push(3)	–	[5, 3]
len(S)	2	[5, 3]
S.pop( )	3	[5]
S.is_empty( )	False	[5]
S.pop( )	5	[ ]
S.is_empty( )	True	[ ]
S.pop( )	"error"	[ ]
S.push(7)	–	[7]
S.push(9)	–	[7, 9]
S.top( )	9	[7, 9]
S.push(4)	–	[7, 9, 4]
len(S)	3	[7, 9, 4]
S.pop( )	4	[7, 9]
S.push(6)	–	[7, 9, 6]
S.push(8)	–	[7, 9, 6, 8]
S.pop( )	8	[7, 9, 6]

Tabela mostrando uma série de operações e seus efeitos numa pilha *S*, inicialmente vazia, de objetos inteiros

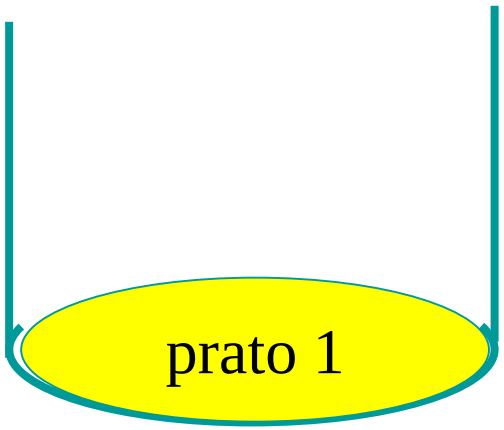
Pilhas
Operações na Pilha (cont.)



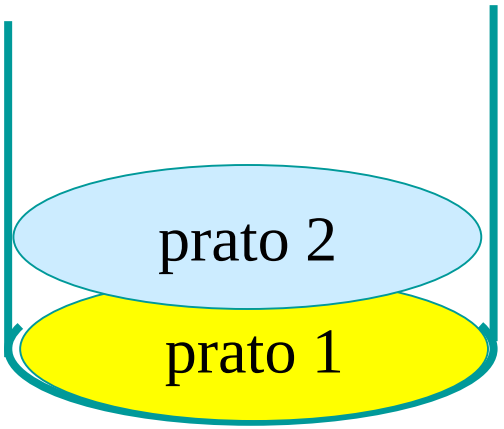
Pilhas

Operações na Pilha – Push em uma Pilha de Pratos

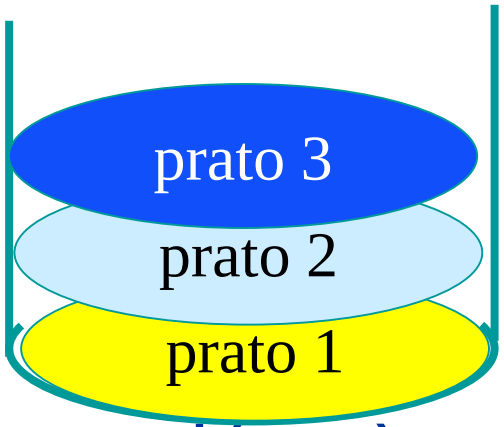
Inicialmente,
a pilha *S* está vazia



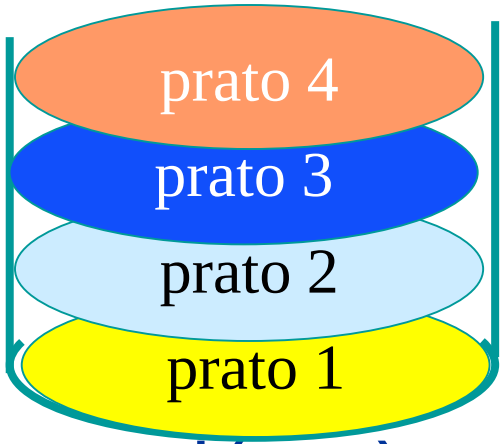
push(prato)



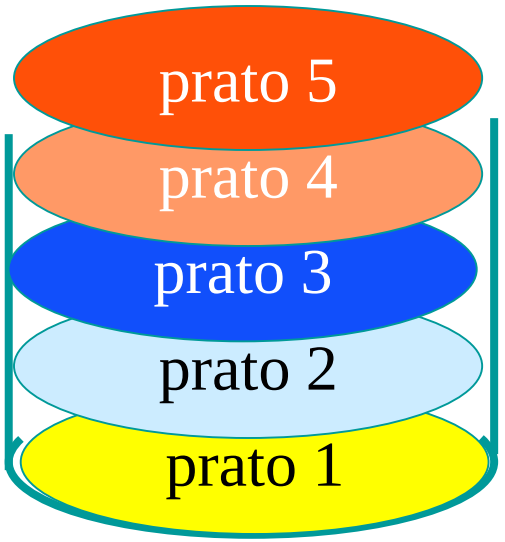
push(prato)



push(prato)



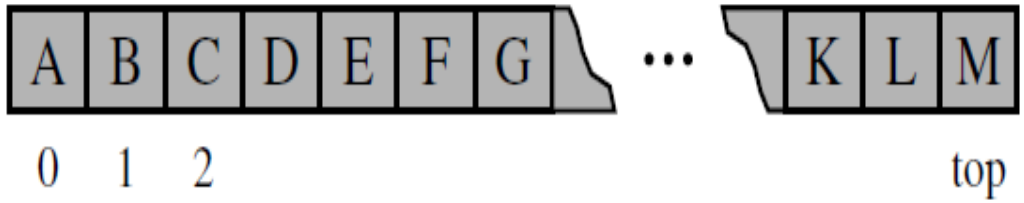
push(prato)



push(prato)

Pilhas
Implementação de Pilha em Array

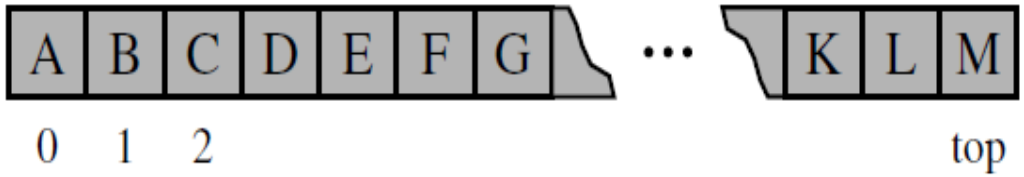
- A implementação de uma pilha por intermédio de um *array* é simples e eficiente utilizando-se a classe *list*
  - já há o suporte para a adição de um elemento no final da lista com o método *append*
  - e a remoção do último elemento da lista com o método *pop*, de modo que é natural alinhar-se o topo da pilha com o final da lista



Implementing a stack with a Python list, storing the top element in the rightmost cell.

Pilhas
Implementação de Pilha em Array (cont.)

- Entretanto
  - A classe *list* inclui comportamentos adicionais que quebram a abstração que o ADT Pilha representa
  - As terminologias dos métodos são diferentes do padrão



Implementing a stack with a Python list, storing the top element in the rightmost cell.

- Torna-se interessante desenvolver uma classe Pilha, utilizando o padrão *Adapter*

Pilhas
Implementação de Pilha em Array (cont.)

- O padrão *Adapter* se aplica a qualquer contexto onde se queira alterar uma classe de modo que seus métodos correspondam aos de uma relacionada, mas diferente classe
  - Define-se uma nova classe de modo que ela contenha uma instância da classe existente como um campo escondido e então implementa-se cada método da nova classe utilizando os métodos da variável de instância escondida. No contexto do ADT Pilha se tem então

Stack Method	*Realization with Python list*
S.push(e)	L.append(e)
S.pop( )	L.pop( )
S.top( )	L[−1]
S.is_empty( )	len(L) == 0
len(S)	len(L)

Realization of a stack S as an adaptation of a Python list L.

Pilhas
Implementação de Pilha em Array (cont.)

- Uma questão a ser analisada na implementação do ADT Pilha é o que fazer quando se chama o método *pop* ou *top* quando a pilha está vazia
  - Há a ocorrência de um erro, mas qual tipo de erro?
  - Quando o método *pop* é chamado em uma lista Python vazia, é levantada uma exceção do tipo *IndexError*, já que listas são sequências baseadas em índices
  - Esta escolha não parece apropriada para pilhas, já que elas supostamente não são baseadas em índices
- Pode-se então definir uma nova classe de exceção

```
class Empty(Exception):
    """Error attempting to access an element from an empty container."""
    pass
```

Pilhas
Implementação de Pilha em Array (cor

```
S = ArrayStack( )              # contents: [ ]
S.push(5)                      # contents: [5]
S.push(3)                      # contents: [5, 3]
print(len(S))                  # contents: [5, 3];      outputs 2
print(S.pop( ))                # contents: [5];         outputs 3
print(S.is_empty( ))           # contents: [5];         outputs False
print(S.pop( ))                # contents: [ ];         outputs 5
print(S.is_empty( ))           # contents: [ ];         outputs True
S.push(7)                      # contents: [7]
S.push(9)                      # contents: [7, 9]
print(S.top( ))                # contents: [7, 9];      outputs 9
S.push(4)                      # contents: [7, 9, 4]
print(len(S))                  # contents: [7, 9, 4];   outputs 3
print(S.pop( ))                # contents: [7, 9];      outputs 4
S.push(6)                      # contents: [7, 9, 6]
```

```
1  class ArrayStack:
2    """LIFO Stack implementation using a Python list as underlying storage."""
3
4    def __init__(self):
5      """Create an empty stack."""
6      self._data = [ ]                       # nonpublic list instance
7
8    def __len__(self):
9      """Return the number of elements in the stack."""
10     return len(self._data)
11
12   def is_empty(self):
13     """Return True if the stack is empty."""
14     return len(self._data) == 0
15
16   def push(self, e):
17     """Add element e to the top of the stack."""
18     self._data.append(e)                   # new item stored at end of list
19
20   def top(self):
21     """Return (but do not remove) the element at the top of the stack.
22
23     Raise Empty exception if the stack is empty.
24     """
25     if self.is_empty( ):
26       raise Empty('Stack is empty')
27     return self._data[-1]                  # the last item in the list
28
29   def pop(self):
30     """Remove and return the element from the top of the stack (i.e., LIFO).
31
32     Raise Empty exception if the stack is empty.
33     """
34     if self.is_empty( ):
35       raise Empty('Stack is empty')
36     return self._data.pop( )               # remove last item from list
```

Code Fragment 6.2: Implementing a stack using a Python list as storage.

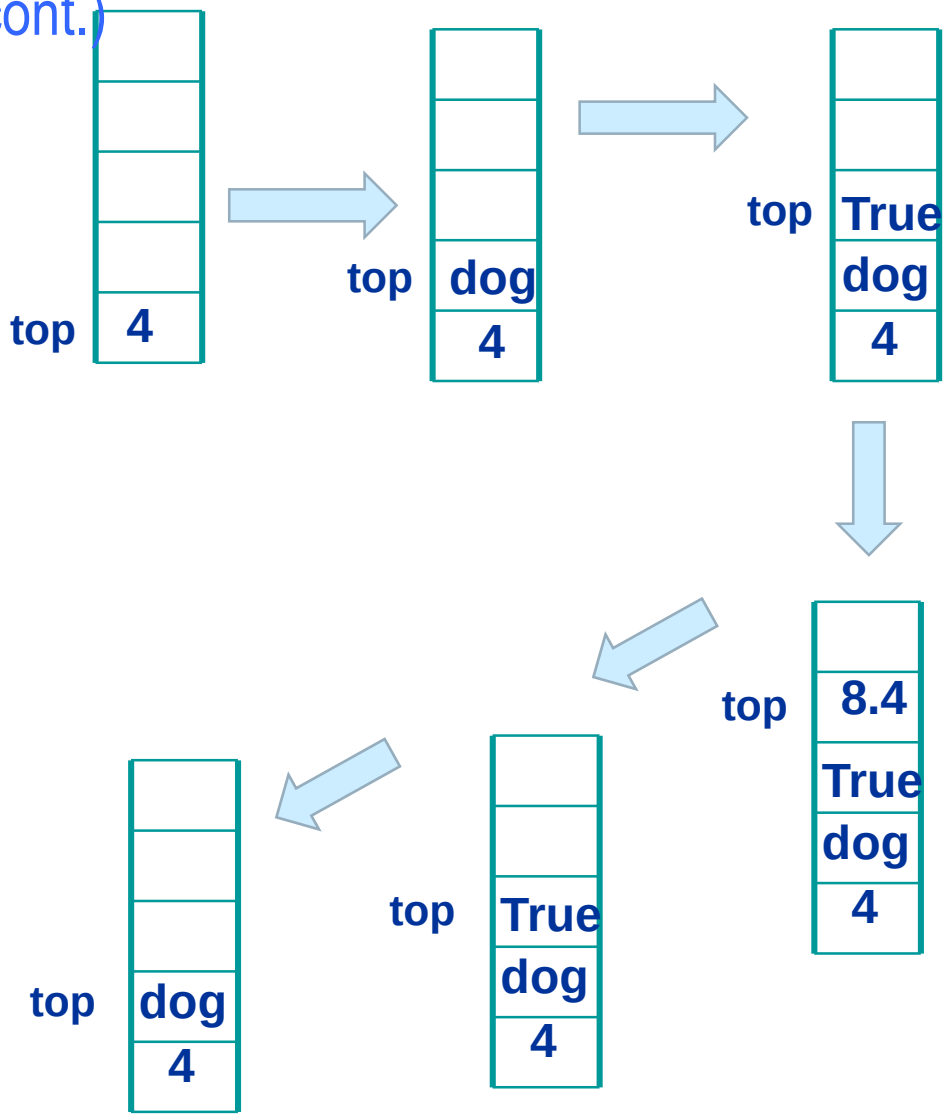
Pilhas

Implementação em Pilha em Array (cont.)

- Code:

```
from Stack import Stack

s = Stack()
print(s.is_empty())
s.push(4)
s.push('dog')
print(s.peek())
s.push(True)
print(s.size())
print(s.is_empty())
s.push(8.4)
s.pop()
s.pop()
print(s.size())
```

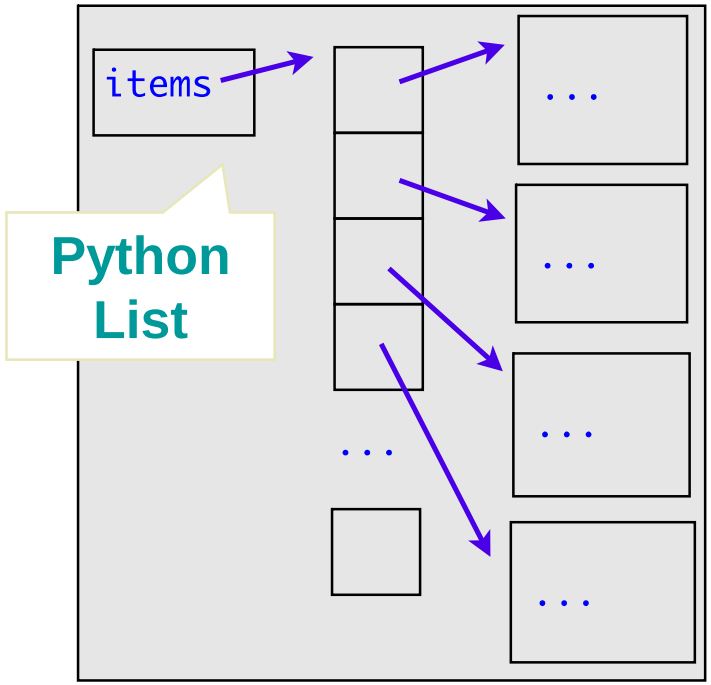


Pilhas
Implementação de Pilha em Array (cont.)

- We use a Python List data structure to implement the stack
  - Remember: the addition of new items and the removal of existing items always takes place at the same **end**, referred to as the **top** of the stack
    - But which “end” is better?

```
class Stack:
    def __init__(self):
        self.items = []
    def is_empty(self):
        return self.items == []
     def size(self):
        return len(self.items)

     def push(self, ...
     def pop(self)
     def peek(self)
```

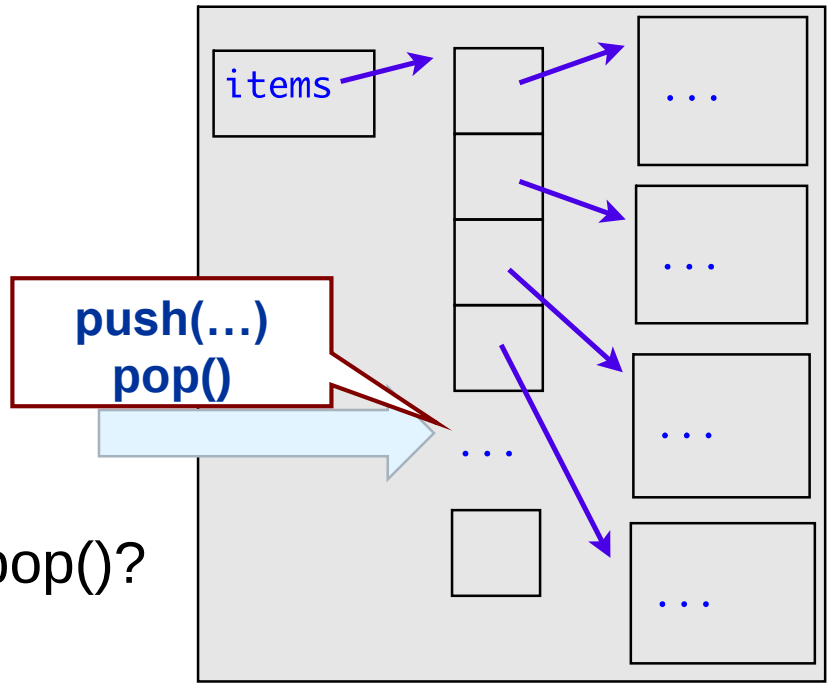


Pilhas
Implementação de Pilha em Array (cont.) – Versão 1

- This implementation assumes that the end of the list will hold the top element of the stack
  - As the stack grows, new items will be added on the **END** of the list. Pop operations will manipulate the same end

```
class Stack:
  ...
  def push(self, item):
    self.items.append(item)

  def pop(self):
    return self.items.pop()
 ...
```



  - What is the Big-O of the push()/pop()?

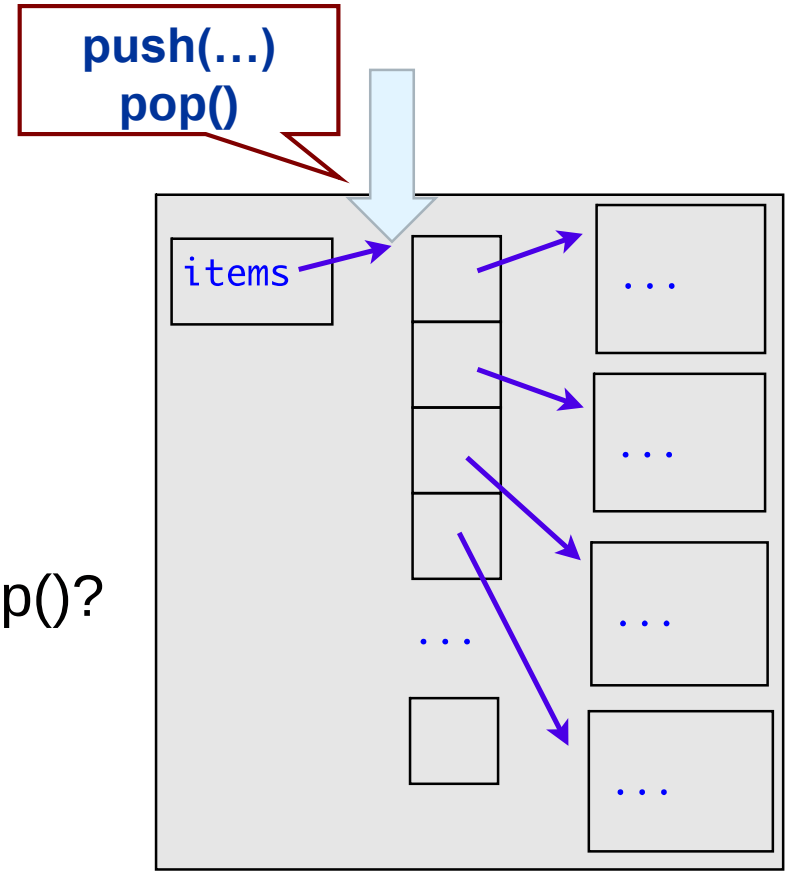
O(1)

Pilhas
Implementação de Pilha em Array (cont.) – Versão 2

- This implementation assumes that the **beginning** of the list will hold the top element of the stack.

```
class Stack:
  ...
  def push(self, item):
    self.items.insert(0,item)

  def pop(self):
    return self.items.pop(0)
 ...
```



- What is the Big-O of the push()/pop()?

O(n)

Pilhas
Pilha Baseada em Array – Desempenho

Método	Tempo
S.push(e)	$O(1)$
S.pop()	$O(1)$
S.top()	$O(1)$
S.is_empty()	$O(1)$
len(S)	$O(1)$

O desempenho de uma pilha implementada em *array*. O uso de espaço é $O(N)$, onde N é o número máximo de elementos que a pilha pode conter, determinado quando a pilha é instanciada. Observa-se que o espaço é independente do numero $n \leq N$ de elementos que estão contidos na pilha num dado momento.

Performance and Limitations

- Performance
  - Let n be the number of elements in the stack
  - The space used is $O(n)$
  - Each operation runs in time $O(1)$
- Limitations
  - The maximum size of the stack must be defined a priori and cannot be changed
  - Trying to push a new element into a full stack causes an implementation-specific exception

Pilhas
Pilhas - ADT Pilha em Python

- Código de implementação da pilha

```
class Stack:
    def __init__(self):
        self.items = []
    def is_empty(self):
        return self.items == []
    def push(self, item):
        self.items.insert(0, item)
    def pop(self):
        return self.items.pop(0)
    def peek(self):
        return self.items[0]
    def size(self):
        return len(self.items)
```

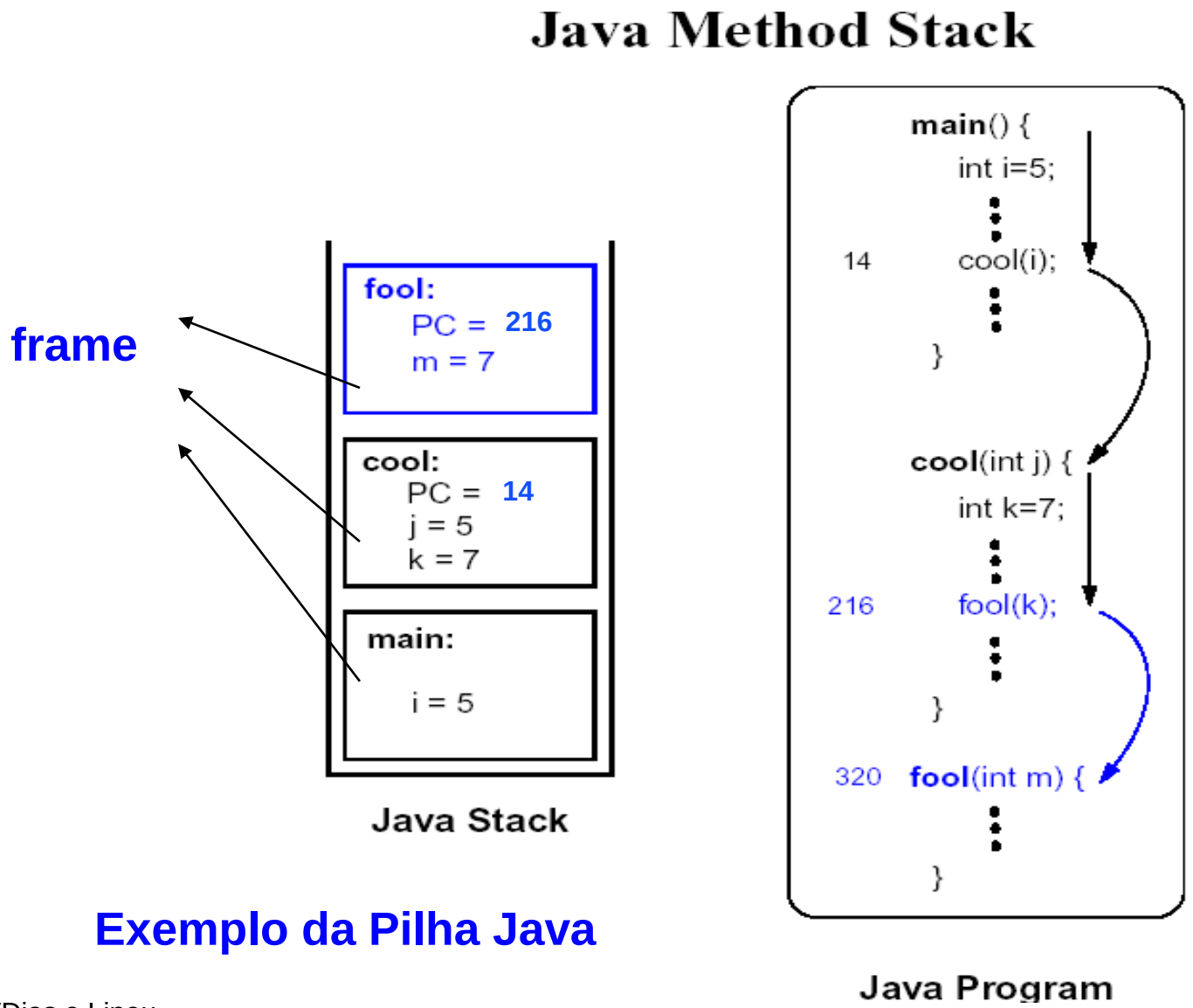
```
s = Stack()
s.push('hello')
s.push('true')
print(s.pop())
print(s.pop())
```

```
============ RESTART:
C:/Python-36/Codigo-Python-
PEDS/Stack1.py ============
true
hello
>>>
```

Pilhas
Máquina Virtual Java (cont.)

- Cada programa executado em Java tem sua própria pilha privada, denominada de pilha de método Java (***Java method stack***) ou pilha Java (***Java stack***), que é utilizada para armazenar informação sobre variáveis locais e outra informações importantes sobre métodos, na medida em que eles são invocados durante a execução do programa.
- Cada vez que um método é chamado ou invocado, ele é empilhado nesta pilha.
- A escolha de uma estrutura como a pilha para esta operação permite ao Java fazer diversas operações úteis como:
  - Efetuar chamadas recursivas a métodos.
  - Imprimir conteúdo da pilha para localizar um erro.

Pilhas
Máquina Virtual Java (cont.)



Exemplo da Pilha Java

Pilhas
Máquina Virtual Java (cont.)

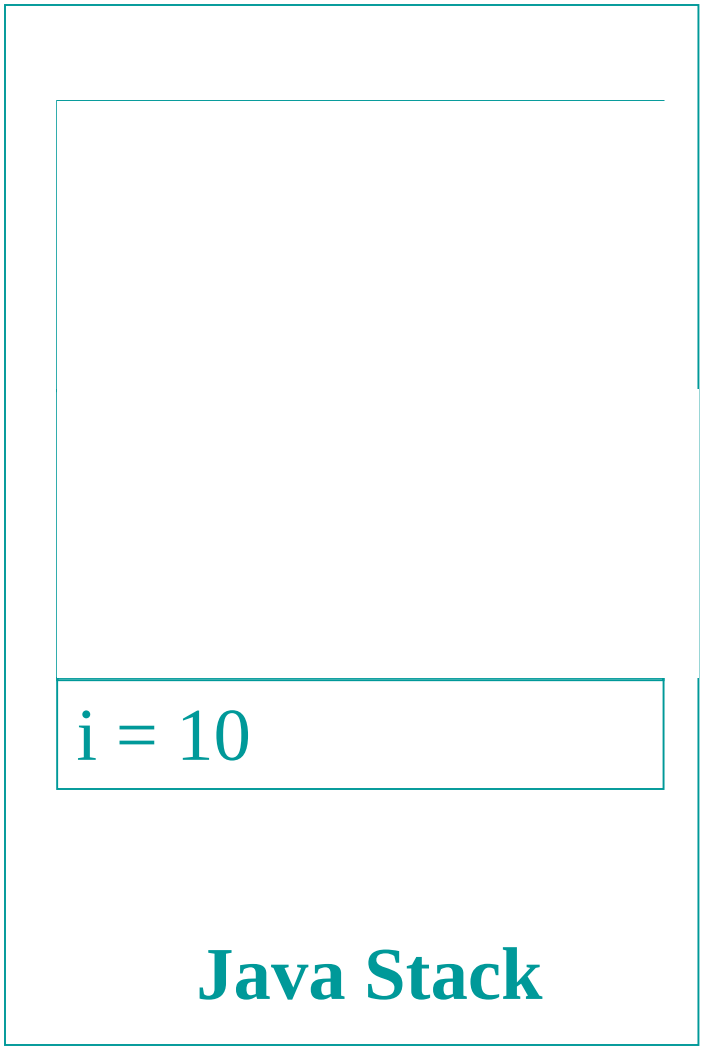
Java Program

```
main() {
   int i = 10;
   . . .
   methodA(i);
}
```
15→

```
methodA(int j) {
   int k = 2 * j;
   . . .
   methodB(k);}
```
93→

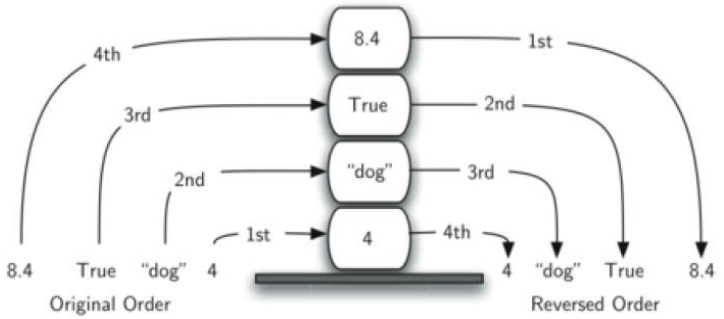
203→
```
methodB(int m) {
   int i = 3 * m;

   . . .}
```



Pilhas
O Uso de Pilhas em Aplicações

- Uma pilha pode ser usada como uma ferramenta para inverter uma sequência de dados
  - Se os valores 1, 2 e 3 forem empilhados em uma pilha nesta ordem, eles poderão ser desempilhados da pilha na ordem 3, 2 e então 1
  - Pode-se querer mostrar dados gravados em um arquivo em ordem crescente em ordem decrescente. Para se fazer isto, lê-se cada linha do arquivo e empilha numa pilha e então se escreve num arquivo na ordem que as linhas são desempilhadas



Pilhas
O Uso de Pilhas em Aplicações (cont.)

- Exemplo de código

```
def reverse_file(filename):
  """Overwrite given file with its contents line-by-line reversed."""
  S = ArrayStack()
  original = open(filename)
  for line in original:
    S.push(line.rstrip('\n'))     # we will re-insert newlines when writing
  original.close()

  # now we overwrite with contents in LIFO order
  output = open(filename, 'w')    # reopening file overwrites original
  while not S.is_empty():
    output.write(S.pop() + '\n')  # re-insert newline characters
  output.close()
```

Pilhas
O Uso de Pilhas em Expressões Matemáticas

- Uma expressão matemática é bem-parentizada (*well-bracketed*) se:
  - para todo parênteses esquerdo **(**, existe um parêntesis direito **)** posterior .
  - para todo parênteses direito **)**, existe um parêntesis esquerdo **(** anterior.
  - a subexpressão dentro de um par de parênteses é, ela mesmo, uma expressão bem-parentizada.
- Exemplos:

$s \times (s - a) \times (s - b) \times (s - c)$ — well-bracketed

$(-b + \sqrt{(b^2 - 4ac)}) / 2a$ — well-bracketed

$s \times (s - a) \times (s - b \times (s - c)$ — ill-bracketed

$s \times (s - a) \times s - b) \times (s - c)$ — ill-bracketed

$(-b + \sqrt{(b^2 - 4ac)} / 2a$ — ill-bracketed

Pilhas
O Uso de Pilhas em Expressões Matemáticas (cont.)

- Algoritmo de parentização

  Para testar se uma expressão é bem parentizada:

  1. Faça a pilha de parêntesis *P* vazia
  2. Para cada símbolo *sym* na expressão (percorrendo da esq. para direita) repita:
     - 2.1 Se *sym* é um parêntesis esquerdo
       - 2.1.1 Adicione *sym* ao topo da pilha *P*
     - 2.2 Se *sym* é um parêntesis direito
       - 2.2.1 Se a pilha *P* está vazia, termine com *false*
       - 2.2.2 Remova o parêntesis do topo de *P* para uma variável *left*
       - 2.2.3 Se *left* e *sym* não são parêntesis complementares, termine com *false*
  3. Termine com *true* se a pilha *P* está vazia, ou *false* caso contrário

Pilhas
O Uso de Pilhas em Expressões Matemáticas (cont.)

- Steps:

Initialise the stack to empty

For every char read

- If it is an open bracket then push onto stack
- If it is a close bracket,
  - If the stack is empty, return ERROR
  - Else, pop an element out from the stack
- if it is a non-bracket character, skip it

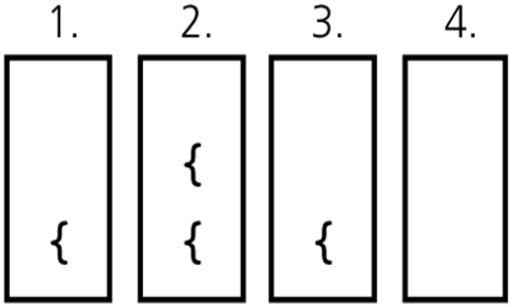
If the stack is NON-EMPTY, ERROR

Pilhas
O Uso de Pilhas em Expressões Matemáticas (cont.)

Input string **Stack as algorithm executes**

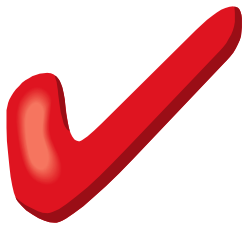
Input string | Stack as algorithm executes

`{a{b}c}`

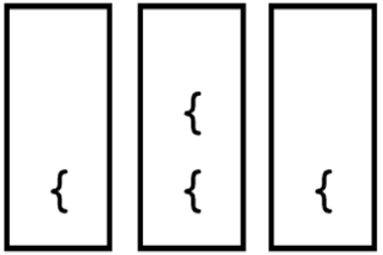


1. push "{"
2. push "{"
3. pop
4. pop

Stack empty ⟹ balanced



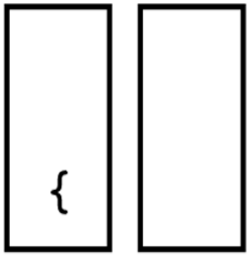
`{a{bc}`



1. push "{"
2. push "{"
3. pop

Stack not empty ⟹ not balanced

`{ab}c}`



1. push "{"
2. pop

Stack empty when last "}" encountered ⟹ not balanced

Pilhas
O Uso de Pilhas em Expressões Matemáticas (cont.)

- E para se avaliar expressões que, além de parênteses, utilizam chaves e colchetes, do tipo
  - 2 + { [F * 4] + [(A – B) / (C – 2)] – 3 } + 4
- Como avaliar se esta expressão é correta, quanto aos parênteses, colchetes e chaves ?

Pilhas
O Uso de Pilhas em Expressões Matemáticas (cont.)

- Steps:

Initialise the stack to empty

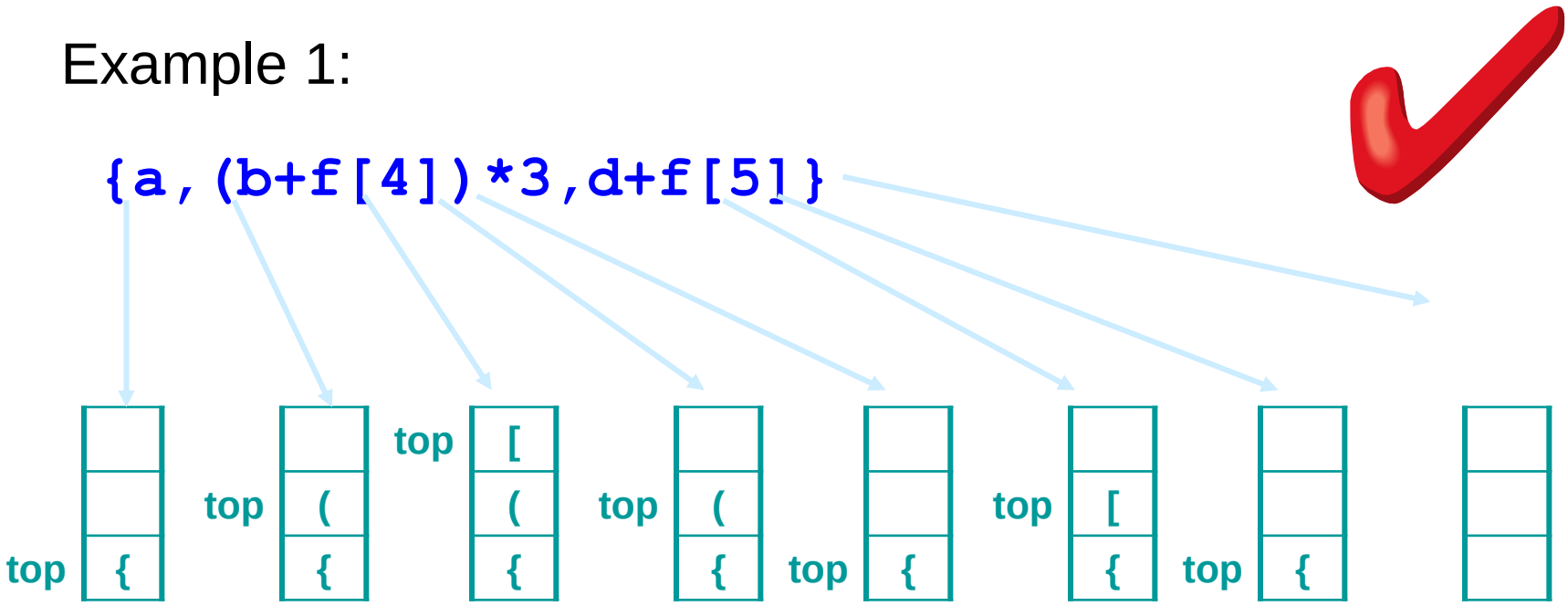
For every char read

- **if it is an open bracket then push onto stack**
- **if it is a close bracket, then**
  - **If the stack is empty, return ERROR**
  - **pop from the stack**
  - **if they don't match then return ERROR**
- **if it is a non-bracket, skip the character**

If the stack is NON-EMPTY, ERROR

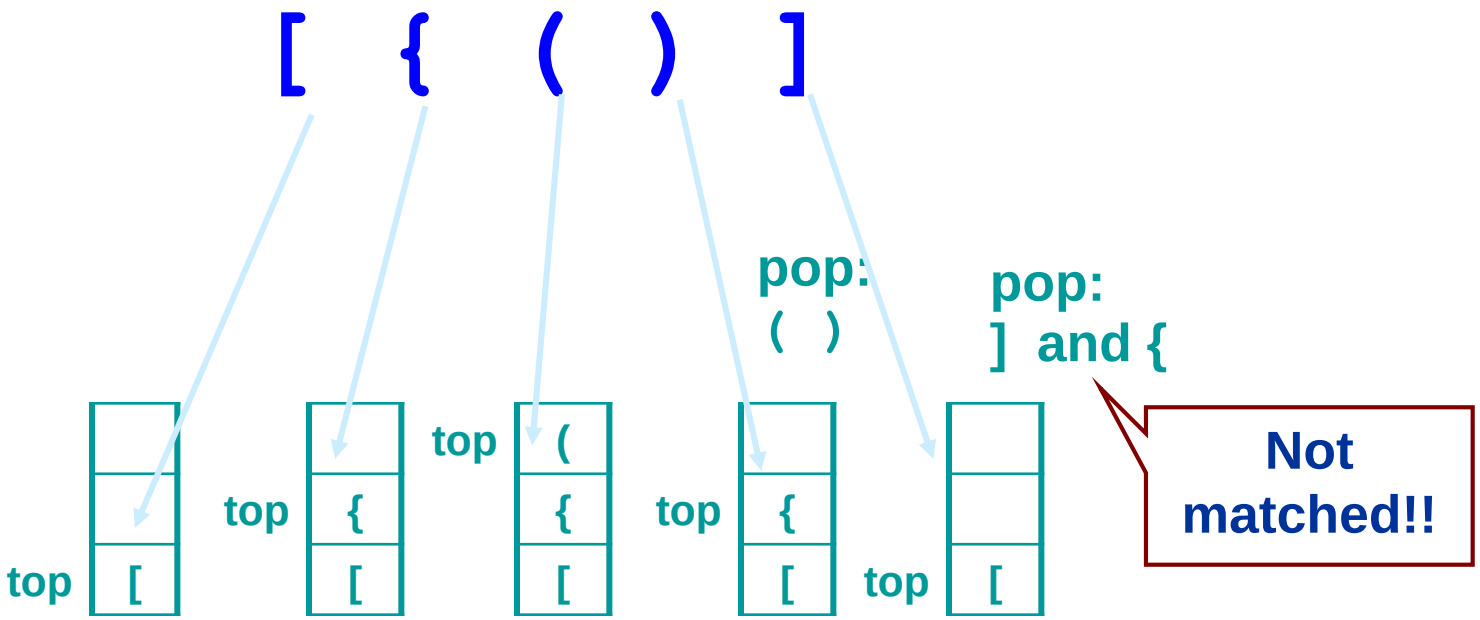
Pilhas
O Uso de Pilhas em Expressões Matemáticas (cont.)

Example 1:



Pilhas
O Uso de Pilhas em Expressões Matemáticas (cont.)

Example 2:



Pilhas
O Uso de Pilhas em Expressões Matemáticas (cont.)

- Seja a seguinte expressão matemática:

$$(5 + 9) \times 2 + 6 \times 5$$

- Para se determinar o valor dessa expressão, primeiro calcula-se a soma 5 + 9 e depois multiplica-se por 2. Então calcula-se o produto 6 $\times$ 5 e este é somado ao resultado anterior para obter-se a resposta final
- Note que a ordem em que as operações são feitas é crucial. É obvio que se as operações não forem realizadas na ordem correta, obtém-se o resultado errado

Pilhas
O Uso de Pilhas em Expressões Matemáticas (cont.)

- Seja a seguinte expressão matemática:

$$(5 + 9) \times 2 + 6 \times 5$$

- A expressão anterior é escrita por meio da notação matemática usual.
- Essa notação é chamada de ***notação infixa***.
- O que distingue essa notação é a maneira na qual as expressões envolvendo os operadores binários são escritas
- Um operador binário é um operador que possui exatamente dois operandos, como por exemplo + e $\times$.
- Na notação infixa, os operadores binários aparecem entre os operandos.

Pilhas
O Uso de Pilhas em Expressões Matemáticas (cont.)

- Seja a seguinte expressão matemática:

$$(5 + 9) \times 2 + 6 \times 5$$

- Uma outra característica da notação infixa é que a ordem das operações é determinada pela precedência do operador. Por exemplo, o operador $\times$ (multiplicação) tem precedência mais alta que o operador + (adição).
- Quando a ordem de avaliação requer uma precedência diferente daquela estabelecida para os operadores, usa-se os parêntesis “(“ e “)”. Uma expressão entre parênteses é avaliada primeiro.

Pilhas
O Uso de Pilhas em Expressões Matemáticas (cont.)

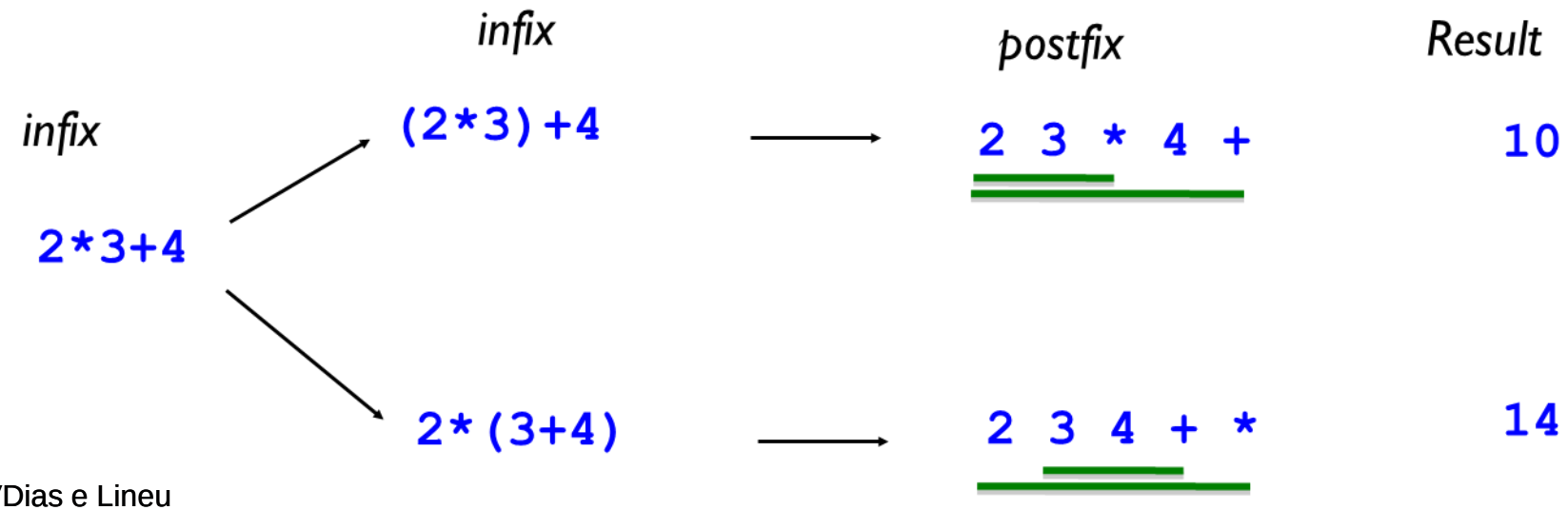
- O lógico polonês *Jan Lukasiewicz* apresentou notações alternativas para a notação infixa que não necessitam do uso de parêntesis e também não requerem uma ordem de precedência dos operadores
- A primeira dessas notações, chamada de ***notação polonesa***, posiciona os operadores antes dos operandos
- Para a expressão (5 + 9) × 2 + 6 × 5, escreve-se nessa notação + × + 5 9 2 × 6 5
- Essa notação é também chamada de ***notação prefixa*** porque os operadores vêm antes dos operandos. Embora a notação prefixa seja absolutamente sem ambiguidade no que diz respeito à ausência de parêntesis, ela não é fácil de ser lida

Pilhas
O Uso de Pilhas em Expressões Matemáticas (cont.)

- Outra forma de notação de *Jan Lukasiewicz* é a chamada ***notação polonesa reversa*** – NPR.
- A expressão (5 + 9) × 2 + 6 × 5 é escrita na forma NPR da seguinte forma: 5 9 + 2 × 6 5 × +
- Essa notação é também chamada de ***notação posfixa*** por razões óbvias, pois os operadores estão posicionados depois dos operandos.
- A notação posfixa, bem como a notação prefixa, não faz uso de parênteses e não requer precedência de operadores. Uma expressão posfixa pode sempre ser escrita sem o uso de parêntesis para expressar a ordem requerida.

Pilhas
O Uso de Pilhas em Expressões Matemáticas (cont.)

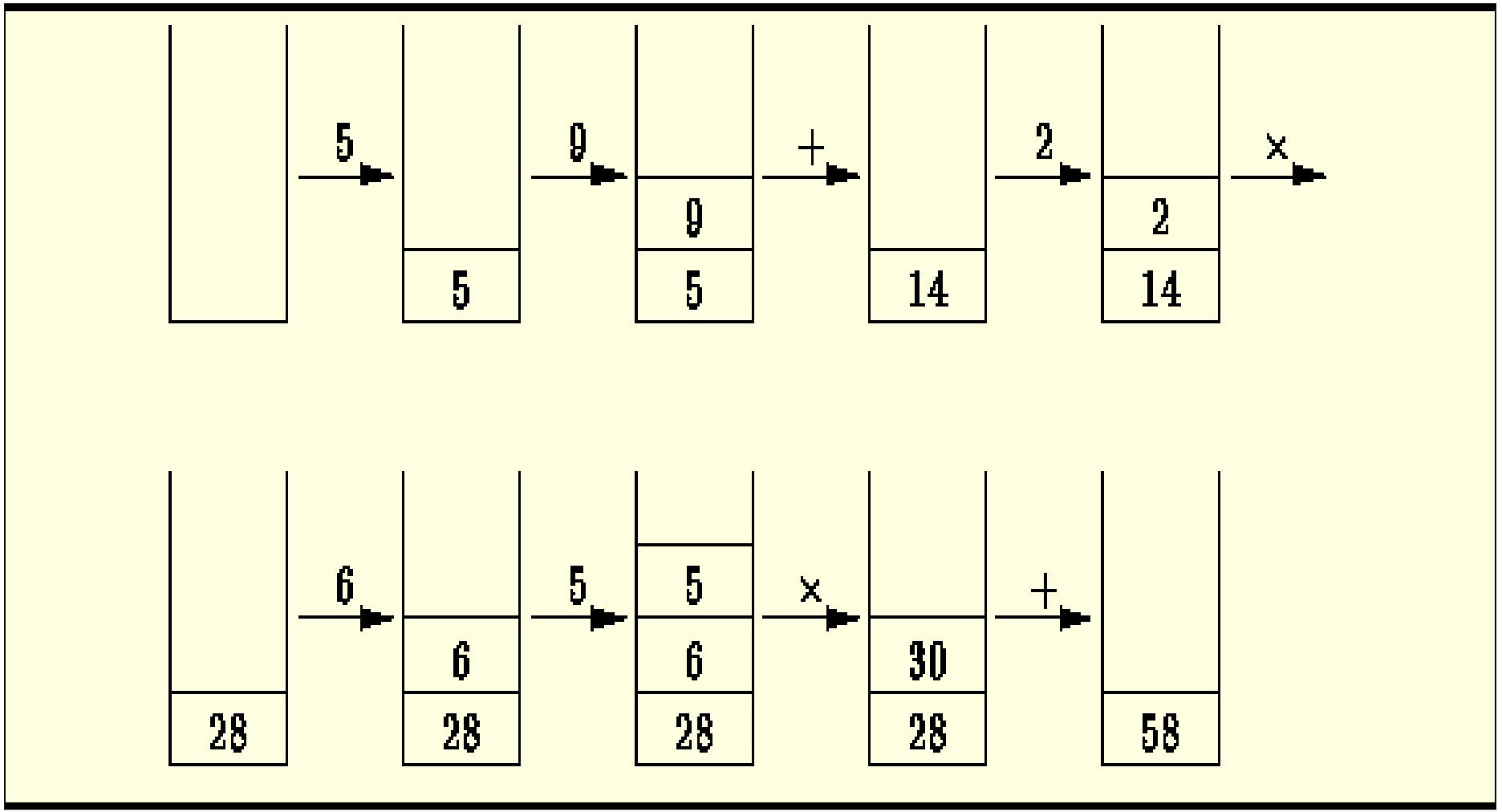
- Por exemplo, a expressão 1 + 2 × 3, na qual a multiplicação é realizada primeiro, é escrita da forma 1 2 3 × +, enquanto a expressão (1 + 2) × 3 é escrita 1 2 + 3 ×
- Sintetizando
  - Infixa - arg1 op arg2
  - Prefixa - op arg1 arg2
  - Posfixa - arg1 arg2 op



Pilhas
Avaliando Expressões Posfixas

- Uma das características mais úteis de uma expressão pósfixa é que seu valor pode ser calculado facilmente com a ajuda de uma pilha de valores
- Os componentes de uma expressão posfixa são processados da esquerda para a direita como se segue:
  - Se o próximo componente da expressão é um operando, o valor do componente é colocado na pilha
  - Se o próximo componente da expressão é um operador, então seus operandos estão na pilha. O número requerido de operandos é retirado da pilha, a operação especifica é realizada e o resultado é armazenado de volta na pilha
- Depois de todos os componentes da expressão terem sido processados dessa forma, a pilha ainda terá um único resultado, que é o valor final da expressão

Pilhas
Avaliando Expressões Posfixas (cont.)



Avaliação da expressão 5 9 + 2 × 6 5 × +, usando-se uma pilha.

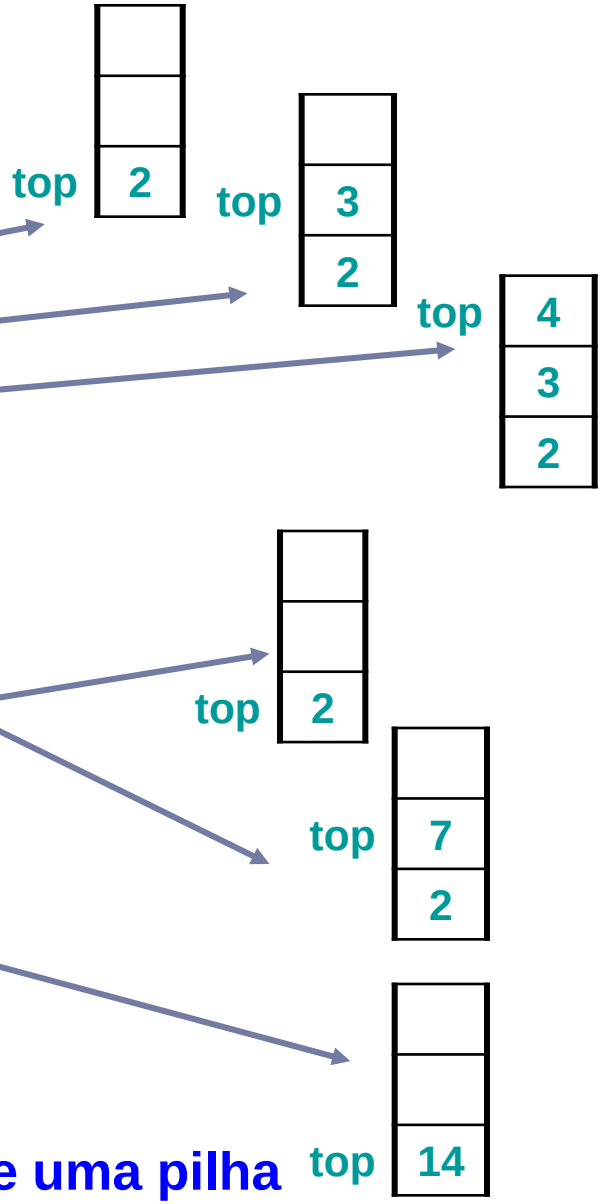
Pilhas
Avaliando Expressões Posfixas (cont.)

Key entered	Calculator action		Stack (bottom to top)
2	push 2		2
3	push 3		2 3
4	push 4		2 3 4
+	operand2 = pop stack	(4)	2 3
	operand1 = pop stack	(3)	2
	result = operand1 + operand2	(7)	2
	push result		2 7
*	operand2 = pop stack	(7)	2
	operand1 = pop stack	(2)	
	result = operand1 * operand2	(14)	
	push result		14

Avaliação da expressão 2 3 4 + * usando-se uma pilha

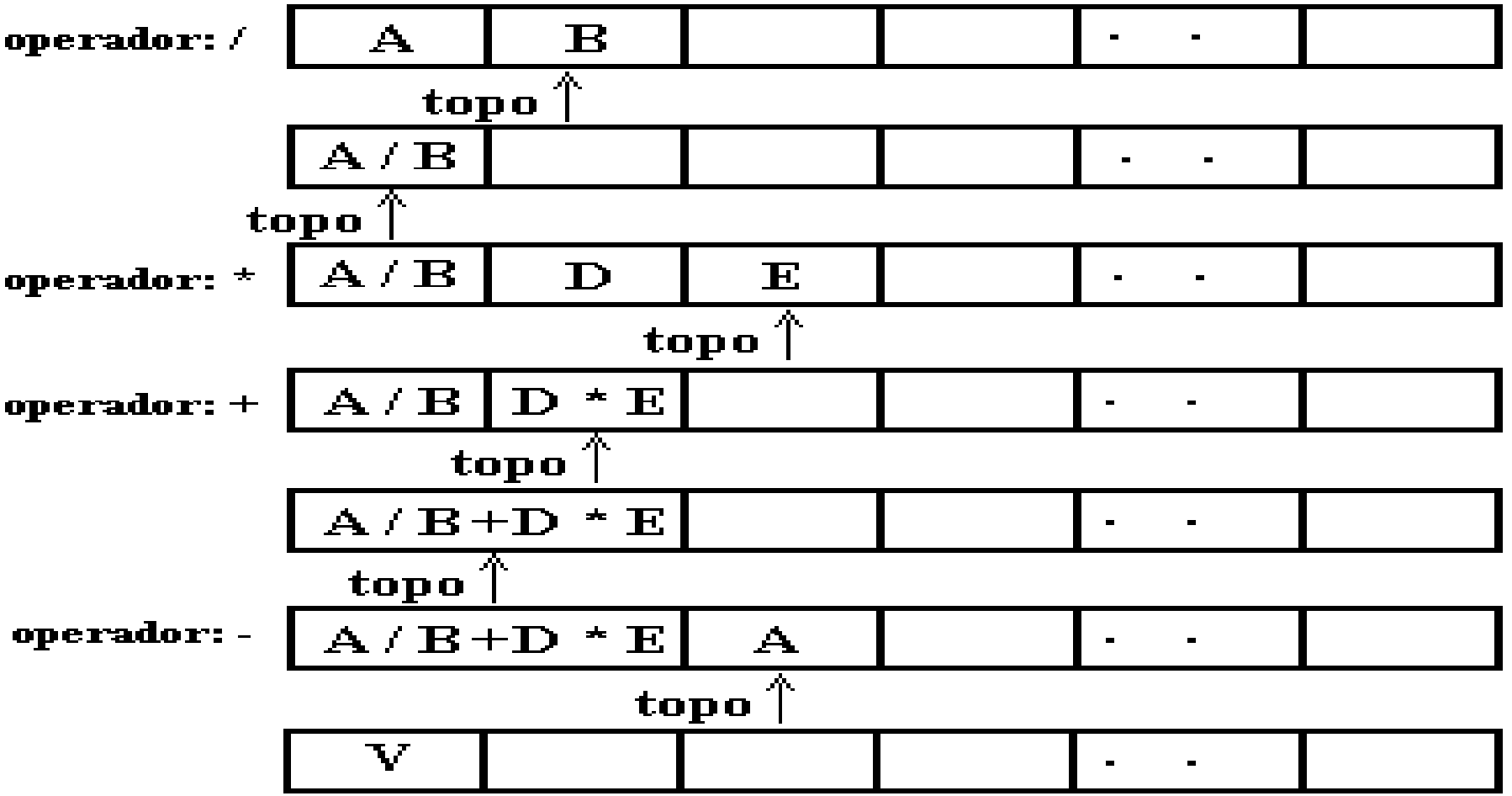
Pilhas
Avaliando Expressões Posfixas (cont.)

key entered	Calculator action
2	s.push (2)
3	s.push (3)
4	s.push (4)
+	arg2 = s.pop () arg1 = s.pop () s.push (arg1 + arg2)
*	arg2 = s.pop () arg1 = s.pop () s.push (arg1 * arg2)



Avaliação da expressão 2 3 4 + * usando-se uma pilha

Pilhas
Avaliando Expressões Posfixas (cont.)



Avaliação da expressão A B / D E * + A - +, usando-se uma pilha

Pilhas
Avaliando Expressões Posfixas (cont.)

- Evaluate: a b c + e f - / *

Read a	Read b	Read c
push a	push b	push c
		c
	b	b
a	a	a

Read +	Read e	Read f	Read -	Read /	Read *
pop c	push e	push f	pop f	pop (e+f)	pop (e+f)/(e+f)
pop b			pop e	pop (b+c)	pop a
add			subtract	divide	multiply
push		f	push	push	push
	e	e	(e - f)		
(b + c)	(b + c)	(b + c)	(b + c)	(b+c)/(e+f)	
a	a	a	a	a	a*((b+c)/(e+f))